

LINUX EXPERT

Frequently asked questions

Linux may give you all the tools you need to achieve your goals, but what if you don't know how to use those tools? Jon Thompson answers your most frequently asked questions

Whether you want to monitor your users' every move, find out who's taking all the processor time or change the colours used by the ls file listing command, there's bound to be a built-in command that can help you achieve your goal. The reason is that Linux is a group effort.

Some commands are compiled programs in languages such as C and C++, but others are actually scripts that anyone can write and distribute. You might even come up with a useful command yourself, which you can release via your own open-source project. If it's popular, you might even have your work included in a future distribution of the operating system.

Although we'll get you started, it's worth digging into the MAN page for each of the commands that we've covered here. Try experimenting with the available command-line switches to see what you can get them to do. Check the 'See also' section at the end of each manual file to find related commands and configuration files. Also, remember that Linux has excellent command-line piping facilities, enabling you to string commands together to create some that are more complex.

Next month, we'll explore regular expressions, showing how you can add even more power to your commands, including some of those covered here.

Trading cases

Q Is there a way of quickly changing the case of a stream of text, or do I have to write a script for a text-processing utility such as sed or awk?

A Try the tr command. This takes piped input and changes all instances of one character into another. It's easier to show than to explain. Consider the following ls output:

```
# ls -l /etc/init.d/s*
-rwxr-xr-x 1 root root 452 2006-04-26
22:39 /etc/init.d/screen
-rwxr-xr-x 1 root root 755 2006-10-06
12:34 /etc/init.d/sendsigs
-rwxr-xr-x 1 root root 585 2006-10-06
12:34 /etc/init.d/single
-rwxr-xr-x 1 root root 4215 2006-10-06
12:34 /etc/init.d/skeleton
-rwxr-xr-x 1 root root 510 2006-10-06
12:34 /etc/init.d/stop-bootlogd
-rwxr-xr-x 1 root root 647 2006-10-06
12:34 /etc/init.d/stop-bootlogd-single
-rwxr-xr-x 1 root root 864 2006-10-20
19:37 /etc/init.d/stop-readahead
-rwxr-xr-x 1 root root 1804 2006-10-06
14:46 /etc/init.d/sysklogd
```

Here's what happens if we pipe the output into tr:

```
# ls -l /etc/init.d/s* | tr a-z A-Z
-RWXR-XR-X 1 ROOT ROOT 452 2006-04-26
22:39 /ETC/INIT.D/SCREEN
-RWXR-XR-X 1 ROOT ROOT 755 2006-10-06
12:34 /ETC/INIT.D/SENDSIGS
-RWXR-XR-X 1 ROOT ROOT 585 2006-10-06
12:34 /ETC/INIT.D/SINGLE
-RWXR-XR-X 1 ROOT ROOT 4215 2006-10-06
12:34 /ETC/INIT.D/SKELETON
-RWXR-XR-X 1 ROOT ROOT 510 2006-10-06
```

```
12:34 /ETC/INIT.D/STOP-BOOTLOGD
-RWXR-XR-X 1 ROOT ROOT 647 2006-10-06
12:34 /ETC/INIT.D/STOP-BOOTLOGD-SINGLE
-RWXR-XR-X 1 ROOT ROOT 864 2006-10-20
19:37 /ETC/INIT.D/STOP-READAHEAD
-RWXR-XR-X 1 ROOT ROOT 1804 2006-10-06
14:46 /ETC/INIT.D/SYSKLOGD
```

We told tr to change all lower-case letters into upper case. We can also have it substitute single characters by providing two patterns. The following example analyses some text and changes all vowels (specified in the first pattern – aeiou) to asterisks (the second pattern):

```
# echo "Mary had a little lamb" | tr aeiou *
M*ry h*d * l*ttle l*mb
```

If you specify more than one letter in the second pattern, tr will change the first letter in the first pattern for the first letter in the second pattern, the second in the first for the second in the second, and so on. If the second pattern doesn't contain as many letters as the first, tr will keep using the last one. For example:

```
# echo "The quick brown fox jumped over
the lazy dog " | tr aeiou 4310
Th3 q01ck br0wn f0x j0mp3d 0v3r th3
14zy d0g
```

We had no character to substitute for u, so tr used the last one it had, which was O. It's useful to know that you can replace longer strings of characters using sed:

```
# echo "This is an example of how to use
this command" | sed 's/this/that/'
This is an example of how to use that
command
```

The sed utility is case sensitive, which is why it didn't make the first substitution and replace 'This' with 'that'.

INTRODUCTION

Linux is a vast environment, and there are lots of useful little commands and utilities lurking in its recesses that can make administering a Linux system much simpler. This month, we examine some of the most powerful in detail. They might not appear to be the easiest tools to use, but the good news is that you probably already have them installed and with our help you'll find them easily.

Jon Thompson
UNIX/Linux management
and security consultant



linux@computershopper.co.uk

Seeing other users' details

Q Is there an easy way of finding detailed information about individual users, including when they logged in and where they are sitting?

A This is really the job of the finger command. Running the finger command gives you an overview of who is logged in at the current time:

```
# finger
Login   Name    Tty    Idle  Login Time   Office   Office Phone
jon     jon     *:0           Dec 15 13:45 (:0.0)
jon     jon     pts/0       Dec 15 13:54 (:0.0)
winston winston pts/1      4    Dec 15 13:54 3rd Floor 555-1212
```

You can get more detailed information about an individual user by specifying their name:

```
# finger winston
Login: jon  Name: Winston Smith
Directory: /home/winston  Shell: /bin/bash
Office: 3rd Floor Mintrue Building Airstrip 1
Office Phone: 555-1212
Home Phone: 555-1313
On since Fri Dec 15 13:54 (GMT) on pts/1 from :0.0
      42 seconds idle
No mail.
Plan: To write a secret diary and thereby defy Big Brother!
```

When you set up a user on the system using a graphical-management tool, you may be offered fields to fill in such as telephone number and other details. These are stored in the /etc/passwd file, specifically in fields 6, 7 and 8, as follows:

```
winston:x:1000:1000:Winston Smith,3rd Floor Mintrue
Building Airstrip 1,555-1212,555-1313,:/home/winston:
/bin/bash
```

The plan information comes from the file called .plan, which is stored in the user's home directory.

The amount of data that you can obtain for an individual user is extensive if you set up each field correctly, so you will need to consider any privacy issues before you actually start. If you have a large network to administer and you need to find phone numbers and office locations, then knowing where your users are located is going to save you considerable time.



▲ If you need to find phone numbers and office locations, then knowing where your users are located will save you lots of time

Checking for duplicates

Q Is there an easy way for me to check a file to see if it contains any duplicate lines?

A The uniq command provides a surprisingly powerful way either to report or suppress duplicates in a file. The default output is simply a stream of lines, where only one instance of any duplicate appears. To explain further, consider the following content of a text file:

```
This is a file containing
two duplicate lines
two duplicate lines
But you see only one of them.
```

Running uniq on this file will give the following output:

```
# uniq file1.txt
This is a file containing
two duplicate lines
But you see only one of them.
```

You can indicate how many duplicates were found with the -c switch:

```
# uniq -c file1.txt
1 This is a file containing
2 two duplicate lines
1 But you won't see them.
```

You can also suppress all instances of duplicate lines with the -u switch:

```
# uniq -u file1.txt
This is a file containing
But you see only one of them.
```

The -d switch has the opposite effect:

```
# uniq -d file1.txt
two duplicate lines
```

Uniq has some useful tricks up its sleeve for pattern matching. Let's suppose you have a file in which each line starts with a unique five-digit code that is followed by some text. You want to ensure that there are no duplicates of the codes, although duplicates of the text are allowed. You can use the -w argument followed by the number of characters that uniq should check for duplicate strings. So in our example we would type:

```
# uniq -d -w 5 file2.txt
```

The inverse of this is the -s switch. This skips the number of characters specified and compares the rest of the line. You can also have uniq ignore the case of the characters it compares by using the -i switch.

There is a shortcoming to uniq, however. If we change the content of the file to read:

```
This is a file containing
two duplicate lines
two duplicate lines
But you see only one of them.
two duplicate lines
```

The output will be:

```
# uniq file1.txt
This is a file containing
two duplicate lines
But you see only one of them.
two duplicate lines
```

The reason is that uniq doesn't buffer the output to check the whole file for duplicates, only those appearing on consecutive lines.

Instant messaging



What do I do if I want to send a message to one or all of the users on my system?



If you run a system that lets other users log in, you may want to communicate with them quickly. The most basic way is by using the wall command:

```
# echo 'This system is not for
playing games on' | wall
```

All terminals get the following message:

```
Broadcast Message from root@jon-
desktop
(/dev/pts/0) at 15:23 ...
This system is not for playing
games on
```

You can also broadcast the content of a file to all users:

```
# wall warning_about_playing_
games.txt
```

Only root can use wall to send the content of a file. If a normal user attempts this, he'll receive the following message:

```
$ wall file1.txt
wall: will not read file1.txt - use
stdin.
```

You can also send a message to a specific user, using the write command:

```
# write jon
write: jon is logged in more than
once; writing to pts/2
Hello there jon.
Kindly stop playing games on this
system.
```

User jon will receive the following:

```
Message from root@bigserver-desktop
on pts/0 at 15:27 ...
Hello there jon.
Kindly stop playing games on this
system.
```

If jon wants to write back, he can use the write command. Stop people interrupting you with write messages by using the mesg command:

```
# mesg n
```

Turn your ability to receive messages back on with mesg y. If you use mesg without an argument, it returns its status (y if others can send you write messages, n if not):

```
# mesg
is y
```

The mesg setting prevents users receiving messages sent by the wall command, unless they come from the root account.

Who's been logging in?



How can I tell quickly who has been logged into the system and from where they have logged in? How can I get a history of who has logged into the system, and what current users are doing?



A command called last shows which users last logged into the system. To do this, it searches through the file called /var/log/wtmp. This file stores a history of logged-in users, plus changes in runlevel. It has a few useful switches. Running last -n 5 will display just the last five lines. Using the -i switch displays the IP addresses from which remote users have logged in. Local logins are designated as 0.0.0.0. The -x switch will include shutdown and runlevel changes, too. Here's a sample output using these switches:

```
# last -n 7 -i -x
jon pts/1 0.0.0.0 Sun Dec 17 10:45 still logged in
jon pts/0 0.0.0.0 Sun Dec 17 10:42 still logged in
jon :0 0.0.0.0 Sun Dec 17 10:40 still logged in
runlevel (to 1v1 2) 0.0.0.0 Sun Dec 17 10:40 - 11:45 (01:05)
reboot system boot 0.0.0.0 Sun Dec 17 10:40 - 11:45 (01:05)
runlevel (to 1v1 6) 0.0.0.0 Sat Dec 16 20:53 - 10:40 (13:46)
runlevel (to 1v1 2) 0.0.0.0 Sat Dec 16 20:53 - 20:53 (00:00)
reboot system boot 0.0.0.0
```

The -t switch allows you to specify the earliest date and time from which the command should display its output. This lets you control the output in terms of time rather than number of entries. The format is -t YYYYMMDDHHMMSS so you could use 'last -t 20061215120000' to display entries from midday on 15th December 2006. The last command is related to the w command, which will give you a quick indication of who's logged in at the moment:

```
# w
13:42:32 up 3:02, 3 users, load average: 0.04, 0.14, 0.16
USER TTY FROM LOGIN@ IDLE JCPU PCPU WHAT
jon :0 - 10:40 ?xdm? 2:58m 0.30s x-session-manager
jon pts/0 :0.0 12:39 17.00s 0.21s 7.14s gnome-terminal
jon pts/1 :0.0 10:45 0.00s 0.29s 7.14s gnome-terminal
```

This gives a quick snapshot of the current users and general activity levels. In addition, the first line of output shows the current time, the length of time since the last boot, the current number of users and the load averages on the processor for the past minute, five minutes and quarter of an hour.

Changing priorities



Please can you tell me how to change the priority of my system's tasks?



Two commands take care of this task: nice and renice. The scheduler at the heart of the kernel allocates processor time to processes based on their priority value. On an Ubuntu system, the default priority is 17. Using nice enables you to increase or decrease this priority value:

```
# nice -10 vi file.txt
```

The -10 part doesn't mean minus 10. The hyphen indicates a command-line switch. The number following it is the adjustment to the priority of the process. The command above will launch the vi command with an increased priority of 27. To launch a command with a lower than usual priority, enter a minus number as follows:

```
# nice --10 vi file.txt
```

This reduces the priority to 7. If you have a processor-hungry script, for instance, you can lower its priority using nice and leave it running

in the background without it slowing down the whole system.

Once running, you can also adjust the priority of the process with the renice command, though only root can do this for processes owned by other users:

```
# renice --10 6567
```

This will drop the priority of process 6567 by 10. As root, you can also change the priority of all a user's processes:

```
# renice -u jon -p --10
```

The -u switch gives the name of the user and -p indicates the desired priority for his currently running processes.

Some systems interpret the values passed to nice and renice to set the priority in different ways. On most systems, the lower the nice or renice value, the faster the process will go, and vice versa. Some systems take the priority values as an absolute; others add it or take it away from the default or current value. It's a good idea to consult the system's MAN page for these commands before you use them, to prevent having an adverse effect on your running system.

Run a command repeatedly

Q Is there a way to execute a command repeatedly, such as monitoring the contents of a directory, without having to keep re-running the command or writing a script?

A Linux contains a handy utility to do this called `watch`. If you only want to watch a log file as entries are added, you can use another command called `tail` with the `-f` switch. For example, to follow the progress of a `syslog` file you could type this to view the contents of the log file and see additional entries as soon as they are made:

```
# tail -f /var/log/syslog
Dec 17 10:46:21 jon-desktop syslogd 1.4.1#18ubuntu6: restart.
Dec 17 10:46:21 jon-desktop anacron[4435]: Job 'cron.daily' terminated
Dec 17 10:46:21 jon-desktop anacron[4435]: Normal exit (1 job run)
Dec 17 11:06:21 jon-desktop -- MARK --
Dec 17 11:17:01 jon-desktop /USR/SBIN/CRON[6207]: (root) CMD ( run-parts --
report /etc/cron.hourly)
Dec 17 11:46:21 jon-desktop -- MARK --
Dec 17 12:06:21 jon-desktop -- MARK --
```

If you wanted to watch the contents of a directory change, you would have to use `watch`. Running `ls -lha` will display a snapshot of the current directory's contents. Typing the following will provide you with a view that is refreshed every two seconds:

```
# watch ls -lha
```

It continues to display the directory until you press `Ctrl-C` to exit. The `watch` command normally refreshes its display every two seconds, but the `-n` switch can be used to change the interval. For example, '`watch -n 1 ls -lha`' shows one-second snapshots of the directory.

Another useful switch is `-d`. This displays the changes that have occurred to the output of the command being controlled by `watch` since the last snapshot was taken. It displays these changes in inverse video, so that you can tell at a glance what has changed.

▲ 'Watch -n 1 -lha' shows one-second snapshots of the directory

Capturing and piping

Q Is there a way to capture the output of a command to a file and also pipe it into other commands for further processing?

A Indeed there is. The `tee` command will insert the equivalent of a T-junction into your command line, sending one copy of its output to the standard output (so that it can be piped into another command), and another copy to a file.

There are a number of times when you might want to do this. For instance, you may want to take the output of a command and both process it and store it in a file for logging purposes.

Here is a basic example of the `tee` command in action:

```
# ls -l | tee /tmp/out.txt
```

If you inspect the content of the file `/tmp/out.txt`, you will see that it contains the output of the `ls` command, which also appeared on the screen.

It is important to remember when using `tee` that it will, by default, overwrite the content of the file you specify if it already exists. You can prevent it doing this with the `-a` switch, which causes it to append to the end of the file instead:

```
# ls -l | tee -a /tmp/out.txt
```

Let's take a more complex example. Suppose we want to use `tee` to take the output of `ls` and make it go to both `/tmp/out.txt` and also to pipe it into another command, such as `sort`. Here's how:

```
# ls -l | tee /tmp/out.txt | sort
```

You can also have `tee` ignore interrupt signals, such as pressing `Ctrl-C`, using the `-i` switch. In this case, you will have to kill the process using the `kill` command:

```
# kill -9 <pid>
```

If `tee` doesn't respond, try using `kill -15`, which is the most deadly of the kill signals.

Comparing files

Q How do I check for differences between two files other than by scanning them line by line myself?

A This sounds like a job for the `diff` command. Consider the following two files:

file1:

```
Now is the time for all good men
to come to the aid of the party.
Now is the winter of our discontent
Made glorious summer by this son of York.
```

file2:

```
Now is the time for all good men
to come to the aid of the party.
Now is the autumn of our discontent
Made glorious summer by this son of York.
```

They look similar, but `diff` indicates differences:

```
# diff file1 file2
3c3
< Now is the winter of our
discontent
---
> Now is the autumn of our
discontent
```

This output confirms that there are differences between the files and shows the lines containing the differences. Lines preceded with a `<` relate to file1 and those starting with a `>` relate to file2. In our example, the word 'winter' in file1 has become 'autumn' in file2.

The `diff` utility's output can be cryptic at times because its output was designed to be read by a computer rather than humans. The output forms a kind of script. The line containing `3c3` is to tell other utilities to change line 3 in one file to line 3 in the other, thus converting either file1 to be file2 or vice versa.

Let's add another line to the end of file2, containing 'Computer Shopper is the best'. Running `diff` again gives us the following output:

```
# diff file1 file2
3,5c3,5
< Now is the winter of our
discontent
---
> Now is the autumn of our
discontent
> Computer Shopper is the best
```

`Diff` indicates that it has identified lines in the second file that are different to those in first.

Although it is a complex program, `diff` serves as a good starting point when comparing two files, especially if they are long. Even if you don't understand its output, knowing where differences lie can save you a lot of eyestrain.

Next month

REGULAR EXPRESSIONS

Add power to the commands covered here